Algorithms PA 3 Report

B11901123 電機三 張甡源

Data Structure

1. Edge Storage

   vector<Edge> edges;

   - Uses a vector to store Edge structs
   - Each Edge struct contains:
     - source vertex
     - destination vertex
     - edge weight
2. Adjacency List

   vector<vector<int> > adjacencyList;

   - Vector of vectors
   - Outer vector size = number of vertices
   - Inner vector size = neighboring vertices of that vertex
   - Used for cycle detection
3. Disjoint Set

   class DisjointSet {
      vector<Node> nodes;
      ...
   }

   - The graph size is stored in vertexCount
   - Graph type (directed/undirected) is stored in graphType
   - Each Node contains a parent pointer
   - Do path compression
   - Do union and find operations

Algorithm

Undirected Graph (Similar to Reversed Kruskal's algorithm)

1. Sort edges by descending weight
2. For each edge:
   - If vertices already connected in disjoint set: remove edge and add weight to total
   - If not connected: keep edge and merge vertices in disjoint set
3. Create Maximum Spanning Tree (connected, acyclic)

Directed Graph (Undirected graph implementation + cosideration of removed edges)

1. Edges removed from undirected phase
2. For each removed edge:
   - If weight negative: keep it removed

- If weight positive:
  - Add back to graph
  - Check for cycles using DFS
  - If create cycle: keep it removed
  - If no cycle: add back to tree and subtract weight from total
3. Create weakly connected, acyclic tree and minimizes total weight of removed edges

<u>Time Complexity</u> (E: number of edges, V: number of vertices)

1. undirected graph:
   - C++ library sort => O(ElogE)
   - Initialization of disjoint set => O(V)
   - For O(E) edges, findRoot and merge=> $E \times O(\boldsymbol{\alpha}(V)) = O(E\boldsymbol{\alpha}(V))$

   Total => $O(E\log E + V + E\boldsymbol{\alpha}(V)) \approx O(E\log E + V)$
2. directed graph:
   - Undirected graph process => O(ElogE + V)
   - Iterate through all removed edges => O(E)
   - For O(E) edges, add edge to adjacency list and check for cycles using DFS => $O(E\times (E + V)) = O(E^2 + EV)$

   Total => $O(E\log E + V + E + E^2 + EV) = O(E^2 + EV)$

Results

Case	type	Removed edges' total weight
public_case_0	d	5
public_case_1	u	21
public_case_2	u	-3330
public_case_3	u	-21680
public_case_4	d	0
public_case_7	d	-10515
public_case_8	d	-71075

Findings

1. Two-phase approach for directed graphs
   - Phase 1 uses modified Kruskal's Algorithm
   - Phase 2 applies directed-specific processing
   - Shows how classic algorithms can be adapted and combined to solve complex problems!

2. Data Structure Optimization:

- Initial implementation used simple adjacency matrix with $O(V^2)$ space
- Then tried linked list which was memory efficient but slow for access
- Finally optimized to current version using:
  - Vector-based adjacency lists for $O(V+E)$ space efficiency
  - Disjoint sets with path compression for near $O(1)$ operations
  - Combined approach provides optimal balance of space and time complexity
- Structure improvement of space-time trade-offs